Structural Differences Between Two Graphs through Hierarchies

Daniel Archambault

Structural Differences Between Two Graphs through Hierarchies

Daniel Archambault*
INRIA Bordeaux Sud-Ouest

ABSTRACT

This paper presents a technique for visualizing the differences between two graphs. The technique assumes that a unique labeling of the nodes for each graph is available, where if a pair of labels match, they correspond to the same node in both graphs. Such labeling often exists in many application areas: IP addresses in computer networks, namespaces, class names, and function names in software engineering, to name a few. As many areas of the graph may be the same in both graphs, we visualize large areas of difference through a graph hierarchy. We introduce a path-preserving coarsening technique for degree one nodes of the same classification. We also introduce a path-preserving coarsening technique based on betweenness centrality that is able to illustrate major differences between two graphs.

Index Terms: H.5.0 [Information Systems]: Information Interfaces and Presentation—General G.2.2 [Mathematics of Computing]: Discrete Mathematics—Graph Algorithms

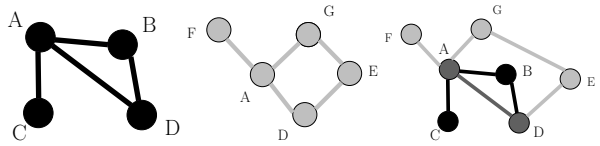
1 INTRODUCTION

A **difference map** is a single graph $G_{\text{diff}} = (N_{\text{diff}}, E_{\text{diff}})$ constructed from the union of two input graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$. This graph encodes all of the differences between the node and edge sets of G_1 and G_2. Figure 1 shows an example.

If the one-to-one correspondence between the nodes of the graphs needs to be computed, the problem is equivalent to subgraph isomorphism which is known to be NP-complete [7]. However, if N_1 and N_2 have a labeling, such that a node in G_1 is the same as a node in G_2, if and only if, their labels are equal, then computing a difference map can be done efficiently. Such graphs exist in practice, especially in the area of dynamic graph drawing. For example, in a computer network, where nodes are servers and edges are connections between those servers, each server has a unique IP address. These IP addresses can be used to determine correspondences between the nodes and edges of G_1 and G_2 which are snapshots of the network at two different dates, demonstrating how the network evolved during that time.

In previous work, animation has been used to show how nodes and edges are added and removed from the graph. If a node is the same in both graphs, its position in the layout is preserved as much as possible and an animation is used to transform G_1 into G_2. This preservation of the mental map is highly dependent on user task and requires compromises with accepted graph drawing aesthetics [21]. An alternative to animation would be to present the differences between G_1 and G_2 in a single drawing. In this drawing, less emphasis would need to be placed on mental map preservation, allowing additional computational resources to be placed on graph drawing aesthetics. A small multiples approach could then be used to demonstrate how the graph evolves over time. **Small multiples** is an information visualization technique that places several images side-by-side in a sequence.

*e-mail: daniel.archambault@inria.fr



(a) Graph G_1 (b) Graph G_2 (c) Difference Map

Figure 1: Example of a difference map between two small example graphs G_1 and G_2 based on the labeling shown in all three subfigures. **(a)** The input graph G_1. **(b)** The input graph G_2. **(c)** The difference map of G_1 and G_2. In this figure, black nodes and edges only appear in G_1, light grey nodes and edges only appear in G_2, and the dark grey nodes and edges appear in both G_1 and G_2.

These difference maps can be very large, possibly on the order of twice the number of nodes and edges in the two input graphs. As these sets can be on the order of tens of thousands of edges, a direct drawing of the difference map suffers from high computational cost and extensive visual clutter. It may be advantageous to consider abstracting away areas of the difference map with similar meaning. In the field of information visualization and graph drawing, graph hierarchies or cluster trees have been used for this purpose. A **graph hierarchy**, or **hierarchy**, is defined as a recursive grouping placed on the nodes in this graph. **Metanodes** are the interior nodes of this hierarchy that contain a subset of nodes and a subset of the edges between those nodes. **Leaves** are the original nodes of the graph. A hierarchy is **path-preserving** if any path in a graph hierarchy level corresponds to one or more paths in the underlying difference map. The two conditions for a path-preserving hierarchy are: the subgraph contained in each metanode is connected and an edge between two metanodes must represent at least one edge in the underlying graph [2]. In this work, metanodes store areas of graph difference, providing the user with an overview of the areas that are common to both G_1 and G_2 or present in only one of these graphs.

The primary contribution of this paper is a method for visualizing the differences between two graphs using a difference map and a graph hierarchy. Using a graph hierarchy allows a difference map to scale to larger graphs, as areas of the graph that the user is not interested in are abstracted away, reducing visual complexity. Our hierarchy creation technique takes into account edges as well as nodes to illustrate both differences. We introduce two path-preserving coarsening techniques. The first technique coarsens away nodes of degree one, while the second coarsens the graph by the magnitude of the change in betweenness centrality of each node. Betweenness centrality is a measure of the number of shortest paths in which the node participates, and, is in a way, a measure of the importance of the node in the graph. Finally, although we defined the properties for a graph hierarchy to be path-preserving elsewhere [2], in this work, we extend the definition of path-preserving to allow nodes of degree one connected to the same node to be coarsened together inside metanodes.

2 PREVIOUS AND RELATED WORK

We categorize related work into two sections. Section 2.1 presents work on dynamic graph drawing. Although our technique is only able to present the differences between two time steps, small mul-

tiples could be used to show how the graph evolves. Section 2.2 presents hierarchy-based graph visualization techniques. We use one of these techniques in order for the visualization of the difference map to be tractable.

2.1 Dynamic Graph Drawing

The field of dynamic graph drawing conveys evolving graphs over time. The problem can be either **offline**, where the full sequence of graphs is known ahead of time or **online**, where the sequence is not known in advance. Our approach would work best in an offline scenario, where all the time steps are available ahead of time.

In offline dynamic graph drawing, Diehl and Görg [9] considered the problem of general dynamic graph drawing by using what they term a *supergraph* which encodes all graphs of the time sequence. Using the supergraph and temporal equivalence classes, they give several layout adjustment strategies in order to update the graph layout between time steps. Erten *et al.* [11] applies a force-directed algorithm to this supergraph as a preprocess in order to derive animations. The work of Brandes *et al.* [5] focuses on creating smooth animations between several graphs in a sequence using spectral graph drawing techniques. An animation sequence which dynamically adds or removes elements from a graph hierarchy is presented in Kumar and Garland [19].

Online dynamic graph drawing started with a restricted set of graphs, mainly directed acyclic graphs drawn in a hierarchical manner [20]. General graphs were considered by Brandes and Wagner [6] where a Bayesian approach, in combination with force-directed techniques, is applied. Some work on drawing orthogonal and hierarchical graphs has been undertaken by Görg *et al.* [16] as well as force-directed methods with weighting functions to evolve the network dynamically while preserving the mental map [13, 14]. Boitmanis *et al.* [3] describe a way to show animations of the evolving Internet. In their approach, trees are removed and drawn with radial clustergrams and the remaining network is drawn using a variant of stress majorization which encourages position stability of nodes between frames of the graph animation.

In all of these systems, animation is used to display graph evolution. There exists some evidence that small multiples may be more accurate than animation for the discovery of trends in evolving data [22]. We are not aware of any experiment run with users to see if this idea extends to dynamic graphs, but it may. Therefore, methods that can be adapted to a small multiples approach for dynamic graph visualization, such as the approach presented here, should be considered more closely.

One could view our approach as applying graph hierarchy techniques to the *supergraph* of Diehl and Görg [9]. However, our technique operates over a pair of time steps rather all of them.

2.2 Hierarchy-Based Graph Visualization

Various techniques exist for visualizing a graph with a superimposed hierarchy including: visualizing the graph and associated hierarchy extruded into the third dimension [10], multi-focal fisheye approaches where metanodes are expanded and viewed in the context of the entire graph [23], topological fisheyes where abstract versions of the graph are presented far away from a focus centre [15], and interactively visualizing hierarchies of small world clusterings [24]. However, all of these techniques require a full layout of the difference map to be computed before visualization of the differences can begin. This full layout is computationally expensive, since our graph sizes are large.

Techniques also exist to draw portions of the graph hierarchy on demand as the user explores the graph [1, 8, 2]. They have the principle advantage that the entire graph does not need to be drawn beforehand. We build on the work described in GrouseFlocks [2] to draw and create our graph hierarchies. However, unlike GrouseFlocks, in this work, we take into consideration the edges of the graph in order to illustrate both node and edge difference.

3 Visualization Approach

In our approach, we visualize the difference map directly using a hierarchy-based graph visualization technique. As input, the algorithm takes two graphs G_1 and G_2. The nodes of G_1 and G_2 have unique labellings L_1 and L_2 such that if an element of G_1 and an element of G_2 have the same label, they correspond to the same nodes in both graphs. The approach is broken down into two main stages:

1. Compute the difference map from G_1 and G_2 using L_1 and L_2. The output of this stage is the difference map G_{diff} with a labeling M, indicating the differences between G_1 and G_2.

2. Using G_{diff} and M, decompose the difference into connected components where all nodes have the same M label and all edges have the same M label.

The first stage, which is computed as a preprocessing step, is discussed in section 3.1. After the computation of the difference map, the visualization of the difference map using hierarchy-based techniques is discussed in section 3.2. The two-level hierarchy uses metanodes to indicate areas of difference in a high-level overview.

3.1 Difference Map Computation

The structural differences between the two graphs G_1 and G_2 are computed based on a pair of unique labellings, L_1 and L_2 on the nodes of the graph. In this first part of the algorithm, we construct a new graph G_{diff} and an output labeling of the nodes and edges, M, that encodes the similarities and differences between G_1 and G_2. Many dynamic graph drawing algorithms compute all or parts of this graph. The algorithm is not a contribution, but rather it is described for completeness.

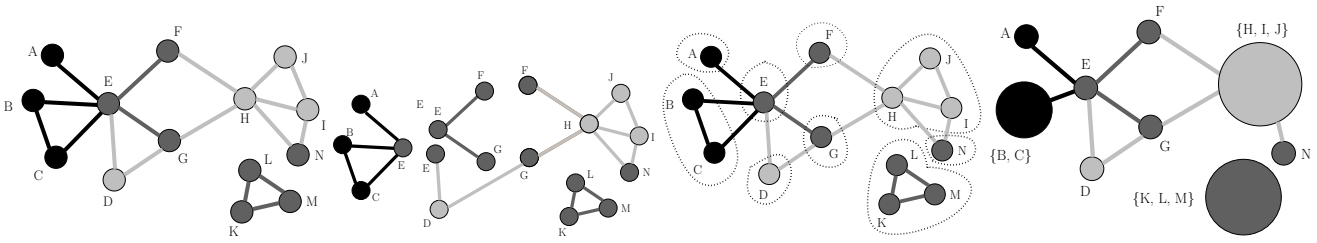
The first part of the algorithm inserts all the nodes of N_1 into G_{diff} and labels them as only belonging to G_1. The nodes are also inserted into a hash table by label. In the second part of the algorithm, the nodes of N_2 are traversed. The algorithm checks to see if the label exists in the hash table. If it does, the node is in both graphs and is labeled as such. If not, the node is inserted into G_{diff} and is labeled as only belonging to G_2.

Edge differences are determined by discovering if the two adjacent nodes in one graph are adjacent in the other. These differences are computed by scanning the edge lists of G_1 and G_2 separately. For each edge of G_1, the algorithm inserts the edge into G_{diff} and labels it as being in G_1 only. The edge is also inserted into a hash table. Once completed, the algorithm scans all the edges of G_2. If either of the nodes belong only to G_2, it is impossible for the edge to exist in both graphs, because one of its adjacent nodes does not exist in both graphs. Therefore, we label the edge as coming from G_2 and insert it into the graph G_{diff}. If the two nodes are in both graphs, there are three possibilities. The first is that the edge is in G_1 only, in which case we inserted the edge in the scan of E_1 and labeled it as such. The second case is that the edge is in both graphs. The algorithm checks the hash table to see if the edge exists, and, if it does, the edge is labeled as belonging to both graphs. Otherwise, the edge is added and is labeled as only belonging to G_2.

If adjacency is encoded using hash tables, the difference map can be computed in $O(N+E)$ time where $N = |N_1| + |N_2|$ and $E = |E_1| + |E_2|$.

3.2 Hierarchy-Based Visualization of the Difference Map

The difference map can be as large as the sum of the sizes of G_1 and G_2. As either G_1 or G_2 can be tens of thousands of nodes and edges [3], visual clutter can be a problem. We, therefore, use



(a) Original Graph (b) Decomposition by Edges (c) Decomposition by Nodes (d) Hierarchy Graph

Figure 2: Steps to creating the hierarchy encoding the node and edge differences in the graph. **(a)** The original difference map with node and edges existing only in the first graph in black, nodes and edges existing only in the second graph in light grey, and nodes and edges existing in both graphs in dark grey. **(b)** Decomposition of the graph by edge difference value. The graph is divided into connected sets of edges of the same type. Nodes belonging to more than one edge set are duplicated. **(c)** Each connected set of edges is further clustered by node difference value. Notice the node N is clustered out of its connected edge set because it appears in both graphs whereas all the other nodes appear only in the second graph. All the nodes participating in more than one edge cluster are placed in their own cluster. **(d)** Final drawing of the hierarchy. Metanodes abstract the graph into clusters where every node has the same difference value and every edge has the same difference value.

graph hierarchies to visualize the structural differences between the graphs.

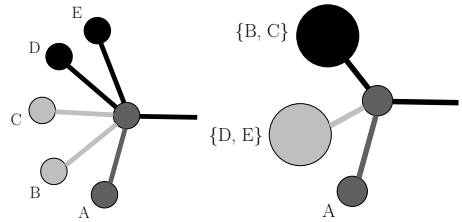
Our visualization technique uses the labeling M to simplify G_{diff} into areas of difference between the two graphs. First, it is important to note that if a node is present in G_1 only, then all adjacent edges to that node must be in G_1 only because the two adjacent nodes of an edge need to both be present for the edge to exist. The same is true for nodes present only in G_2. Therefore, a Grouse-Flocks [2] decomposition on the nodes when the nodes appear in one graph or the other is sufficient. However, for edges whose adjacent nodes present in both graphs, the edge could be present only in G_1, only in G_2, or in both graphs. To address this case, we needed to develop a decomposition technique that takes an edge labeling into account.

Our approach, shown in Figure 2, starts by decomposing the graph into connected clusters of edges with the same M label. If a node belongs to multiple clusters of connected edges in the decomposition, it is duplicated. The clusters of connected edges are subsequently subdivided into clusters of connected nodes with the same M label. First, every node that was duplicated at least once, is placed into its own cluster. Subsequently, each connected set of edges is divided into connected sets of nodes of the same M label. For example, the large connected set of light grey edges is subdivided into six clusters in Figure 2. The nodes E, G, and F exist in their own clusters, because they are in more than one edge cluster. The node N is in its own cluster, because it exists in both graphs and is adjacent only to nodes existing in one graph. Finally, the node D and the nodes H, I, and J are in separate clusters because they are not connected by at least one edge. The result of this algorithm is a path-preserving hierarchy as all subgraphs contained in each metanode are connected and edges will be placed between metanodes if they exist in the difference map.

4 Coarsening Degree One Nodes

GrouseFlocks [2] introduced our definition of a path-preserving hierarchy. The definition in this work required that an edge exists between two metanodes, m_1 and m_2, if and only if, at least one leaf in m_1 and one leaf in m_2 are connected by an edge. Secondly, the definition required that the subgraph contained at each metanode is connected. In this paper, we extend this definition of path-preserving to allow disconnected subgraphs inside a metanode as long as it respects the following condition:

- The subgraph contained in a metanode of a path-preserving hierarchy can be disconnected only if every node in the subgraph is degree one and is connected to the same node s in the hierarchy.



(a) Before Coarsening (b) After Coarsening

Figure 3: Coarsening technique that merges degree one metanodes of the hierarchy together. **(a)** Nodes of the hierarchy are not grouped together, causing a very bushy tree out on the periphery. **(b)** Nodes of the hierarchy are grouped into metanodes according to their M labeling. As all nodes in these metanodes are degree one, no path will ever enter and then exit this metanode in mid-sequence. Therefore, this hierarchy is path-preserving. The coarsening technique can be viewed as clustering the beginnings and endings of paths together.

A path can only begin or end at a node of degree one. As a result, the subgraph contained inside this type of metanode does not need to be connected, because paths do not pass through the metanode. We can view this metanode as a collection of path sources or sinks connected to s in the graph.

In this work, we also restrict the degree one nodes to have the same value in the labeling M computed in Section 3.1. The approach scans every node in the hierarchy once and merges all nodes of degree one that have the same M labeling together into a single metanode. The resultant difference map is simpler for graphs with degree one nodes as shown in Figure 3.

5 Betweenness Centrality Coarsening

Coarsening using the algorithm presented in Section 4 may generate a hierarchy that still contains too many areas of difference. The resultant graph is often large, difficult to read, and is computationally expensive to draw. In this case, our approach adapts by allowing only the *major* differences in graph structure to be shown.

Betweenness centrality [12] is a measure of how often a node lies on shortest paths in the graph. These nodes are often structurally important as they appear on many short, and presumably well used, communication paths. For a node v it's betweenness centrality is computed as follows:

$$BC(v) = \sum_{u \neq v \neq w} \sigma_{(u,w)}(v)$$

where $\sigma_{(u,w)}(v)$ is the number of shortest paths that contains the node v. We do not normalize our betweenness centrality by the number of paths that exist in the graph. Fast algorithms exist to compute the betweenness centrality of all elements of a graph of $|N|$ nodes and $|E|$ edges in $O(|N||E|)$ time [4]. Edge filtering techniques based on betweenness centrality have been used for the visualization of social networks, most recently in Jia *et al.* [18], but not for coarsening a graph hierarchy of graph differences.

Using our approach, the algorithm computes the betweenness centrality for G_1 and G_2 independently before the difference map is computed. When computing the difference map, the absolute value of the difference in betweenness centrality is recorded for each node. For nodes present in only one graph, the value is set to its betweenness centrality in that graph. Low betweenness centrality differences indicate areas of the graph that are stable or of low structural importance. Thus, this metric attempts to emphasize important structural changes in the graph.

The algorithm selects nodes of low betweenness centrality difference and replaces each connected component of the induced subgraph by a single metanode. A node n is selected under three conditions:

1. n is a metanode that contains a subgraph that is identical in both graphs
2. n is not a metanode and has a betweenness centrality difference below a user specified threshold
3. n is not a metanode and appears in both graphs such that:
   - all adjacent edges appear in both graphs
   - all adjacent nodes are below the betweenness centrality threshold

The first condition coarsens away large subgraphs that appear in both graphs contained in metanodes. The second condition coarsens away minor structural changes and areas of stable structure. Nodes with a low difference in betweenness centrality are considered less important by the metric or only had a minor change in structural significance. The third condition coarsens away minor edge changes: sets of edges that appear in both graphs have at least one node below the threshold. Note that any metanode containing a subgraph that appears in only one of the graphs is never coarsened away.

6 Results

In order to test the technique, we ran our algorithm on three sequences of test data. For these results, we used a 3.6GHz Pentium IV with 1GB of RAM running Fedora Core 4. Timing numbers, along with graph and difference map sizes, are presented in Table 1.

The first dataset is the `Threads2` graph sequence used in the work of Frishman and Tal [14]. A movie[1] that shows the evolution of `Threads2` is available from the second author's website for comparison. The dataset consists of evolving discussion threads. Nodes are users discussing a topic and an edge exists between two nodes if one user replied to the posting of the other. A special node, *A_n* where *n* is a number, represents the root of a given topic thread.

The second dataset is two scans of the Opte[2] Internet mapping project. `Opte` consists of two snapshots of a subset of major Internet servers taken on January 11th and January 15th, 2005, respectively. In this graph, nodes are servers and an edge in the graph exists if those servers exchange packets.

The third dataset is the Routeviews[3] dataset used in the work of Boitmanis *et al.* [3]. A movie[4] that shows the full sequence of `Routeviews` evolving over time is available from the website of the research group. The datasets consist of two scans of the Internet at the autonomous system level, one taken in August 2005 and the other taken August 2006. Autonomous systems are typically groups of networks under the same administrative authority. In the graph, there is a node for each autonomous system and an edge between two nodes if the systems are connected.

To draw the resultant hierarchies, we use GrouseFlocks [2]. GrouseFlocks uses algorithms to draw the graphs based on the topological features detected in the graph. Therefore, if the metanode contains a tree, a tree drawing algorithm is used. `Threads2` uses the same set of drawing algorithms specified in the GrouseFlocks paper. However, `Opte` and `Routeviews` use the FM^3 [17] algorithm for subgraphs of unknown topological structure. We use FM^3, as this algorithm is fast and produces reasonable results for large general graphs. Even after the generation of a hierarchy and coarsening, the resultant graphs are still on the order of thousands to tens of thousands of nodes.

6.1 Threads2

In `Threads2`, we are trying to describe the evolution of a set of discussion threads. The results of our approach are shown in Figure 4. Tan nodes are nodes that appeared in both graphs over the time step while purple nodes have just been added. Circular nodes contain subgraphs of the same difference value. Square nodes are the original nodes of the dataset. Neither form of coarsening was used on this small dataset. We notice immediately that there are no deletions from the dataset during the time steps as only two colours are used. Therefore, none of the messages were deleted during these scans. The figures also demonstrate reasonably well what was added in purple. Pinpointing exactly what changed may be harder to do with the video of the evolving graph mentioned above.

In Figure 4(a), we present the difference between time steps 10 and 11 of the dataset with no graph hierarchy. Differences are still coloured using the colour scheme above. In this small example, it is relatively easy to see the differences between time steps without a hierarchy. If the graphs were larger and substantially different, the task would require a more time consuming visual search to detect the differences. On the other hand, Figure 4(b) highlights these differences immediately. We can see that the new discussions were stimulated by the postings of *Locutus465* and *KaiserCSS*. By using an interactive system, such as GrouseFlocks, we could interactively open these two metanodes to see which user responded to this discussion thread.

Figures 4(b) through 4(f) show the evolution of the graph over five time steps. The diagrams show how the discussion in the newsgroup evolves rather easily. In Figure 4(e), the drawing depicts the emergence of a new discussion topic *A_5082*. This topic was started by *Gigahertz 19*. In the next time step, this discussion is continued by *Mazor*. *KristopherKubicki* joins this disconnected discussion with the rest of the newsgroup by replying to at least one new post.

6.2 Opte

The `Opte` difference map encodes the differences between two Internet scans in 2005. The results are presented in Figure 5. Sections of the network that remained the same appear in tan, and new sections that were found on January 15 of 2005 appear in purple. Once again, no parts were deleted as no third colour needed to be used. Therefore, this network only grew.

In Figure 5(a), a FM^3 layout of the entire difference map is presented. General areas of difference are visible in purple, but it is hard to gauge the size of the differences and their locations. Figure 5(b) shows a hierarchy of the differences. The size of the node is mapped to the size of the subgraph. At the center of the diagram, there is a large component of the network that is identical.

[1] `www.ee.technion.ac.il/~ayellet/Movies/OnlineGD.mov`
[2] `www.opte.org`
[3] `www.routeviews.org`
[4] `www.inf.uni-konstanz.de/algo/research/asgraph`

Dataset	Graph 1		Graph 2		Difference Map			B. Centrality		Hierarchy		
	N	*E*	*N*	*E*	*N*	*E*	sec.	sec.	sec.	*N*	*E*	sec.
`Threads2` 10 - 11	62	72	66	76	66	76	0.01	0.02	0.02	4	4	0.14
`Threads2` 11 - 12	66	76	69	80	69	80	0.01	0.02	0.02	12	12	0.22
`Threads2` 12 - 13	69	80	70	83	70	83	0.01	0.02	0.02	10	12	0.20
`Threads2` 13 - 14	70	83	72	87	72	87	0.01	0.02	0.02	10	11	0.24
`Threads2` 14 - 15	72	87	75	91	75	91	0.01	0.02	0.02	6	5	0.14
`Opte` No	35,386	42,387	40,027	47,215	40,027	47,215	3.16	5,389	6,827	2,642	3,132	20
`Opte` 1	35,386	42,387	40,027	47,215	40,027	47,215	3.16	5,389	6,827	1,970	2,460	18.91
`Opte` B.	35,386	42,387	40,027	47,215	40,027	47,215	3.16	5,389	6,827	657	788	14.51
`Routeviews` No	20,432	43,498	23,072	48,891	24,302	58,043	5.61	1,820	2,407	23,656	57,283	104.97
`Routeviews` 1	20,432	43,498	23,072	48,891	24,302	58,043	5.61	1,820	2,407	19,372	52,999	117.32
`Routeviews` B.	20,432	43,498	23,072	48,891	24,302	58,043	5.61	1,820	2,407	3,212	6,773	78.4

Table 1: Size of datasets used in the results and timings for the stages of the creation of the difference map and hierarchy. The **Graph 1** and **Graph 2** columns list the size of the two input graphs in number of nodes and edges. The **Difference Map** column contains the size of the difference map in number of nodes and edges and the time required to compute the difference map in seconds. The **B. Centrality** column lists the time required to compute the betweenness centrality of each input graph in seconds The **Hierarchy** column contains the size of the hierarchy graph in number of nodes and edges and the time required to compute the hierarchy in seconds. *No* next to the dataset name indicates no coarsening. A *1* indicates degree one coarsening. *B.* indicates both degree one and betweenness centrality coarsening were executed. Betweenness centrality thresholds of 600,000 and 2,000,000 were used for `Opte` and `Routeviews`, respectively. The numbers next to `Threads2` indicate the two graphs in the sequence used to create the difference map.

This fact is not immediately evident from the previous drawing. We can begin to see some areas of major growth in the upper right section of the diagram where a few large purple metanodes reside. In Figure 5(c) these changes are still visible. The periphery of the drawing is improved by the degree one clustering, making some of these areas at the fringe of the diagram more visible. Figure 5(d) emphasizes the major edge changes, where major is defined as having a betweenness centrality difference above 600,000. Clutter is greatly reduced as most of the nodes that appeared in both graphs are now in the residing central component. The few edges, between nodes close to this large brown component, are edges adjacent to nodes with a major change in betweenness centrality.

6.3 Routeviews

`Routeviews` tests the limits of our approach. The difference map generated contains tens of thousands of nodes and contains about ten thousand more edges than either of the original datasets. This dataset has both removals and insertions. Once again, areas of the graph which are the same are illustrated in tan. Areas of growth in the graph appear in purple and areas of the graph that disappeared appear in teal. Result images are shown in Figure 6.

Figures 6(a) and 6(b) do not clearly demonstrate the changes in the the graph structure between these two dates. Other than noticing that many of the teal nodes and edges have been placed into components of the hierarchy, we cannot see much, so we need to resort to coarsening. Figure 6(c) demonstrates some of the major differences in the graph structure. A few large areas of the graph that did not change are drawn in tan. There are some areas that were added and taken away in purple and teal, respectively. However, even with degree one coarsening, the changes in the graph are not well depicted. Once we coarsen away areas of the graph that have a betweenness centrality inferior to 2,000,000, as in Figure 6(d), the drawing does a more accurate job of depicting major areas of difference. There is a single large component of stable structure with a few nodes connected to it. A more rapidly changing part of the graph is visible in the lower left portion of the diagram. However, none of these drawings are very clear.

7 Discussion

From the timings shown in Table 1, the only step which requires significant computational time is computing the betweenness centrality on the two input graphs. Currently, we use Brandes' algorithm [4] to compute the betweenness centrality for each node. These times can be reduced by estimating the betweenness centrality at each node, using a technique, such as Jia *et al.* [18], to approximate the value rather than to compute it exactly. Also, metrics and interfaces for selecting a good betweenness centrality threshold should be investigated further.

The difference map approach is best suited for a small multiples approach to graph visualization. A small multiples approach places several images side-by-side in a sequence rather than showing the sequence as an animation. In our work, we would place the hierarchically-decomposed difference maps in a matrix. This approach has the advantage that a user can view all the data at once, but the disadvantage is that all these images require significant space as seen from the size and quantity of images in this paper.

The main advantage of our approach is that it targets the core of dynamic graph visualization: depicting what exactly has changed. Through difference maps, hierarchy generation, and the coarsening techniques presented here, our work depicts differences using spatial position and colour. We think that these differences are easier to notice when depicted in this way, rather than by using animation. However, a user experiment is required to support this hypothesis.

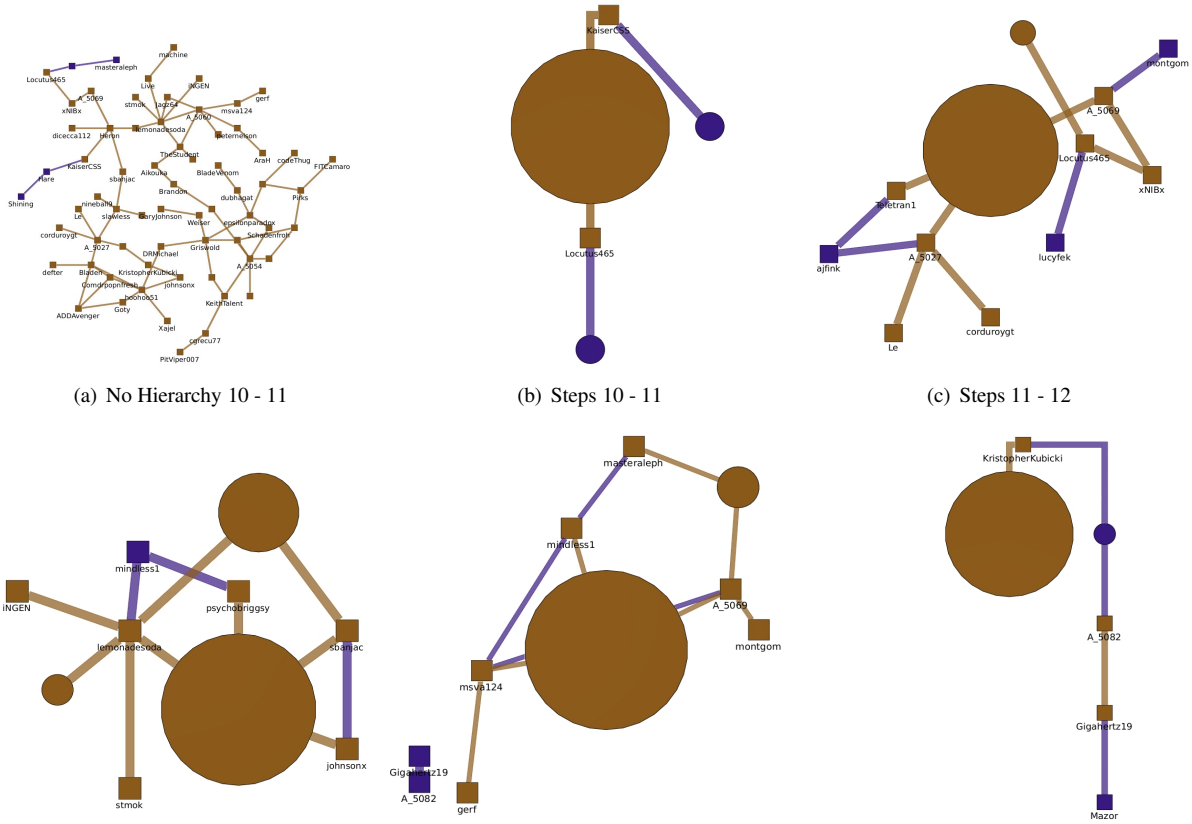
8 Future Work and Conclusions

We plan to run a user study which will hopefully support the hypothesis that the presented difference maps outperform animation when the user would like to discover structural changes in graphs. Such an experiment would also attempt to determine if hierarchy construction on top of the difference map helps users understand how a graph evolves over time.

Currently, our approach does not place metanodes which contain similar graph elements into similar areas of the plane. As spatial position is an important visual cue, ensuring that similar parts of the graph are in similar locations would improve our technique.

It may be an interesting area of future work to illustrate changes in attribute values associated with the nodes and edges of the graph. As an example, in computer networks, an attribute could encode the amount of network traffic passing through an edge between two servers.

In this paper, we have presented a method for visualizing the structural differences between two graphs. Using a graph hierarchy allows the approach to scale, as large areas of the graph that are the



(a) No Hierarchy 10 - 11 (b) Steps 10 - 11 (c) Steps 11 - 12

(d) Steps 12 - 13 (e) Steps 13 - 14 (f) Steps 14 - 15

Figure 4: Figure showing the evolution of the graph sequence `Threads2`. **(a)** The difference between graphs 10 and 11 in the sequence drawn without a hierarchy. **(b) - (f)** Hierarchies demonstrating the differences between a pair of consecutive time steps. Nodes in the graph that appear in both time steps appear in tan. Nodes that have just been added over the last time step appear in purple. Circular nodes contain connected subgraphs of the same difference value and square nodes are nodes of the dataset.

same are abstracted away, reducing visual complexity. Our hierarchy creation technique takes into account edges as well as nodes to illustrate both differences. We introduce two path-preserving coarsening techniques and modify the definition of path-preserving.

Acknowledgements

I would like to thank INRIA Bordeaux Sud Ouest Gravié project for support. I also acknowledge David Auber and Guy Melançon for many interesting discussions on this topic. Thank you Morgan Mathiaut, for helping shoot the figures in the results section. Finally, I would like to thank the anonymous reviewers of GI, as well as Julie, Mari, and David for their useful comments.

References

[1] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. *IEEE Trans. on Visualization and Computer Graphics (Proc. Vis/InfoVis '06)*, 12(5):669–676, 2006.

[2] D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable exploration of graph hierarchy space. *IEEE Trans. on Visualization and Computer Graphics*, 14(4):900–913, 2008.

[3] K. Boitmanis, U. Brandes, and C. Pich. Visualizing internet evolution on the autonomous systems level. In *Proc. of Graph Drawing (GD '07)*, volume 4875 of *LNCS*, pages 365–376, 2007.

[4] U. Brandes. A faster algorithm for betweenness centrality. *J. Math. Soc.*, 25(2):163–177, 2001.

[5] U. Brandes, D. Fleischer, and T. Puppe. Dyanmic sprectral layout of small worlds. In *Proc. of Graph Drawing (GD '05)*, number 3843 in LNCS, pages 25–36, 2005.

[6] U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In *Proc. of Graph Drawing (GD '97)*, volume 1353 of *LNCS*, pages 236–247, 1997.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[8] E. Di Giacomo, W. Didimo, L. Grilli, and G. Liotta. Graph visualization techniques for web clustering engines. *IEEE Trans. on Visualization and Computer Graphics*, 13(2):294–304, March/April 2007.

[9] S. Diehl and C. Görg. Graphs, they are a changing – dynamic graph drawing for a sequence of graphs. In *Proc. of Graph Drawing (GD '02)*, volume 2528 of *LNCS*, pages 23–31. Springer-Verlag, 2002.

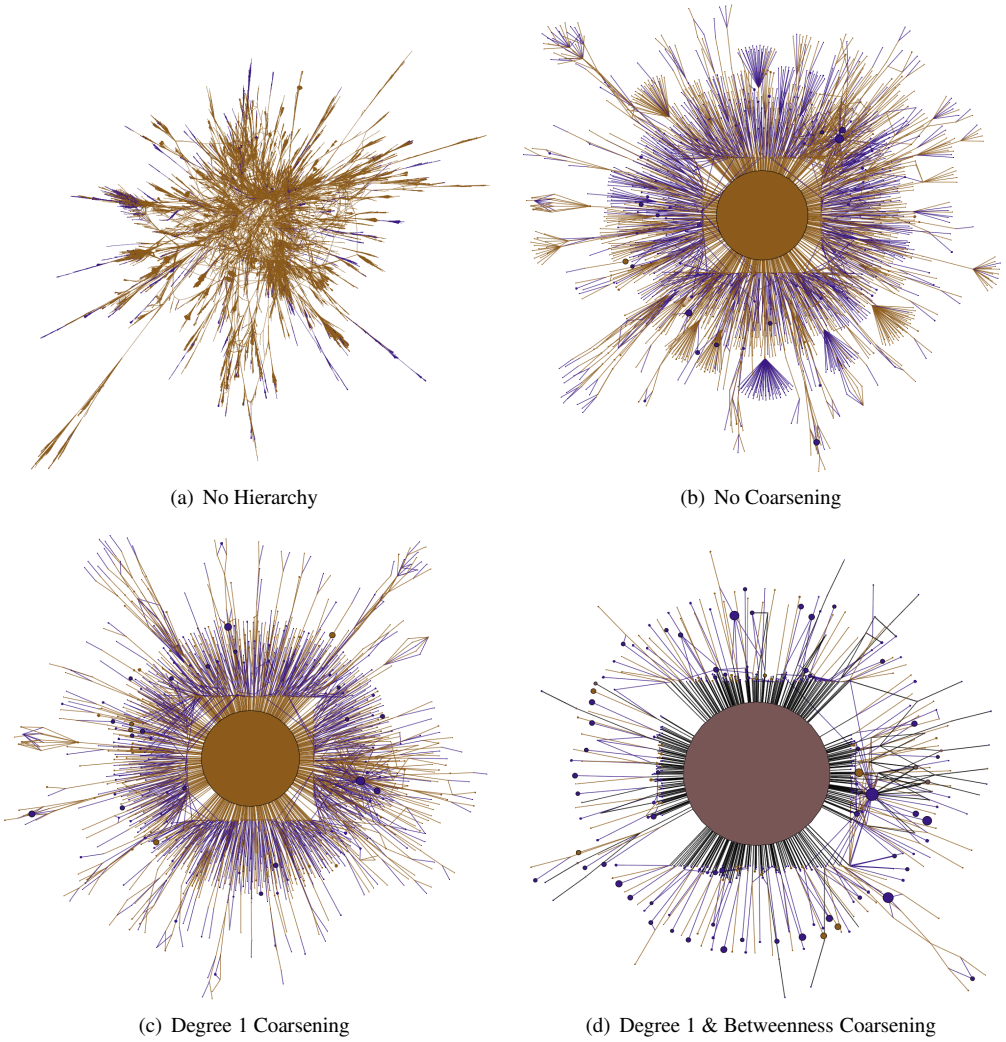
[10] P. Eades and Q. Feng. Multilevel visualization of clustered graphs. In *Proc. Graph Drawing (GD'96)*, volume 1190 of *LNCS*, pages 101–112. Springer-Verlag, 1996.

[11] C. Erten, P. J. Harding, S. Kobourov, K. Wampler, and G. V. Yee. GraphAEL: Graph animations with evolving layouts. In *Proc. of Graph Drawing (GD '03)*, volume 2912 of *LNCS*, pages 98–110. Springer-Verlag, 2003.

[12] L. C. Freeman. A set of measures of centrality based upon betweeness. *Sociometry*, 40(1):35–41, 1977.

[13] Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *Proc. IEEE Symposium on Information Visualization (InfoVis'04)*, pages 191–198, 2004.

[14] Y. Frishman and A. Tal. Online dynamic graph drawing. In *Proc. Eu-*



(a) No Hierarchy (b) No Coarsening

(c) Degree 1 Coarsening (d) Degree 1 & Betweenness Coarsening

Figure 5: Figure showing differences between the January 11th and 15th scans by Opte in 2005. Similarities are in tan and the growth of the network is in purple. No servers were deleted from the scans between these two dates. **(a)** Graph without any hierarchy or coarsening drawn with FM^3. **(b)** Graph with hierarchy but no coarsening. **(c)** Graph with hierarchy and degree one coarsening. **(d)** Graph with degree one and betweenness centrality coarsening. The brown nodes are the components that have a relatively stable or small betweenness centrality. Black edges connect these coarsened components to the rest of the graph. A betweenness centrality threshold of 600,000 was used to generate the betweenness centrality coarsened hierarchy.

rographics/IEEE VGTC Symp. on Visualization (EuroVis'07), pages 75–82, 2007.

[15] E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. *IEEE Trans. on Visualization and Computer Graphics*, 11(4):457–468, 2005.

[16] C. Görg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Proc. of Graph Drawing (GD ' 04)*, volume 3383 of *LNCS*, pages 228–238, 2004.

[17] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 285–295. Springer-Verlag, 2004.

[18] Y. Jia, J. Hoberock, M. Garland, and J. C. Hart. On the visualization of social and other scale free networks. *IEEE Trans. on Visualization and Computer Graphics*, 14(6):1285–1292, 2008.

[19] G. Kumar and M. Garland. Visual exploration of complex time-varying graphs. *IEEE Trans. on Visualization and Computer Graphics (Proc. Vis/InfoVis 2006)*, 12(5):805–812, 2006.

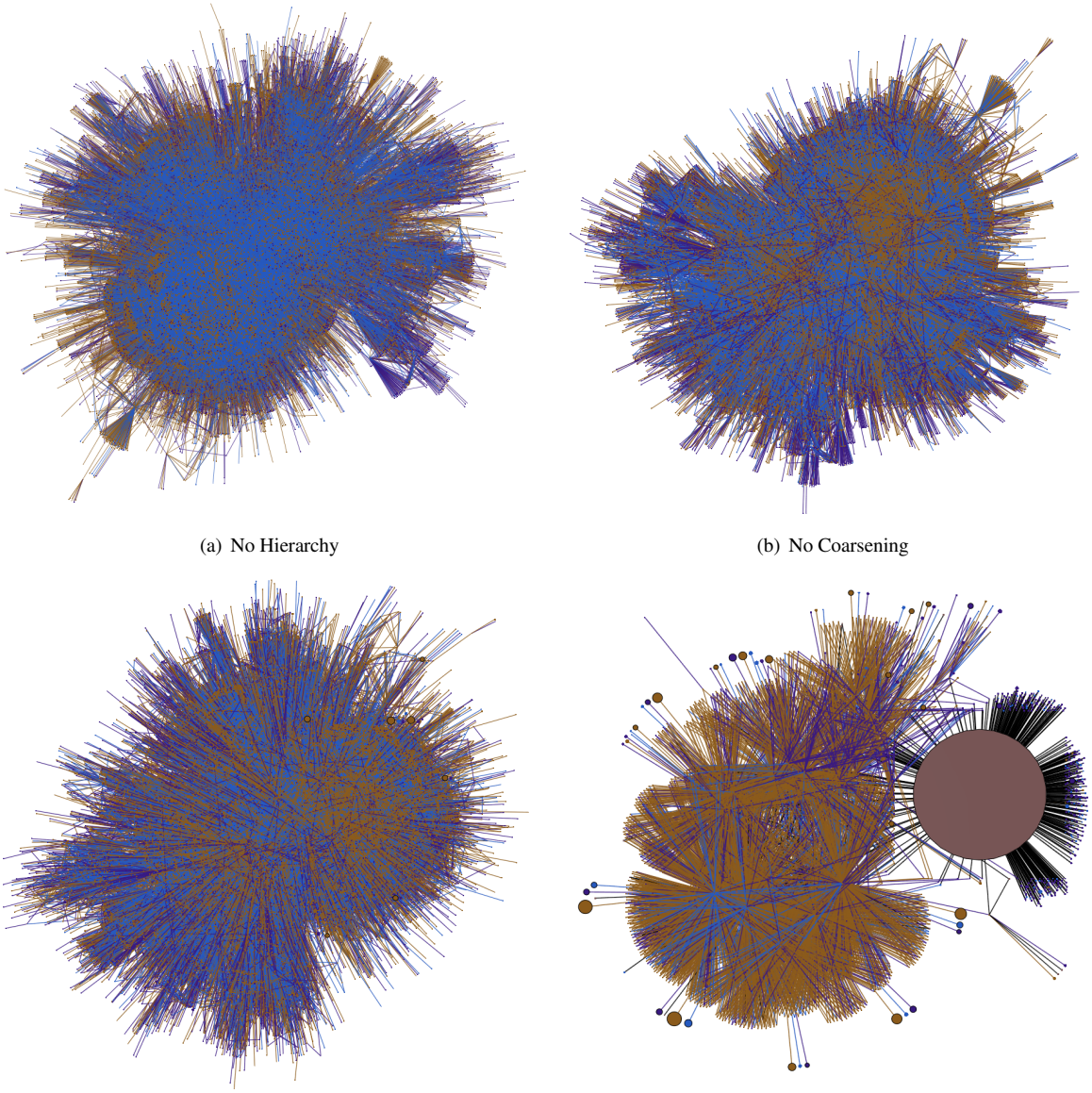
[20] S. North. Incremental layout in dynaDAG. In *Proc. of Graph Drawing*, volume 1027 of *LNCS*, pages 409–418. Springer-Verlag, 1995.

[21] H. Purchase, E. Hoggan, and C. Görg. How important is the "mental map"? – an empirical investigation of a dynamic graph layout algorithm. In *Proc. Graph Drawing (GD '06)*, volume 4372 of *LNCS*, pages 184–195, 2006.

[22] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Trans. on Visualization and Computer Graphics (Proc. Vis/InfoVis '08)*, 14(6):1325–1332, 2008.

[23] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. on Computer-Human Interaction (TOCHI)*, 3(2):162–188, 1996.

[24] F. van Ham and J. van Wijk. Interactive visualization of small world graphs. In *Proc. IEEE Symposium on Information Visualization (InfoVis'04)*, pages 199–206, 2004.



(a) No Hierarchy

(b) No Coarsening

(c) Degree 1 Coarsening

(d) Degree 1 & Betweenness Coarsening

Figure 6: Figure showing differences between two Routviews snapshots taken on August 1st of 2005 and 2006 respectively. Nodes and edges that were the same in both graphs are in tan. Nodes added are in purple and nodes removed are in blue. **(a)** Graph without any hierarchy or coarsening drawn with FM^3. **(b)** Graph with hierarchy but no coarsening. **(c)** Graph with hierarchy and degree one coarsening. **(d)** Graph with degree one and betweenness centrality coarsening. The brown nodes are the components that have a relatively stable or small betweenness centrality. Black edges connect these coarsened components to the rest of the graph. A betweenness centrality threshold of 2,000,000 was used to generate the betweenness centrality coarsened hierarchy.